Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 50 (2015) 264 – 269

Procedia Computer Science

Novel Read Algorithms for Improving the Performance of Big Data Storage Systems

Thirumalaisamy Ragunathan[a], Sudheer Kumar Battula[b], Rathnamma Gopisetty[c], B. RangaSwamy[d], N. Geethanjali[e]

[a]*ACE Engineering College, Hyderabad, India*
[b]*ACE Engineering College, Hyderabad, India*
[c]*GITAM University, Hyderabad, India*
[d]*Prosoft Resources Pvt. Ltd, Hyderabad, India*
[e]*Sri Krishnadevaraya University, Ananthapur, India*

Abstract

Cloud computing systems provide scalable infrastructure to store and process Big Data generated by various organizations. Distributed file system (DFS) is used as the main storage element in a cloud computing system for storing and accessing Big Data. Improving the performance of the read operations in the DFS is one of the important research issues as more frequently the users perform read operations on the DFS and less frequently the write operations. In this paper, we have developed two novel read algorithms for improving the performance of the read operations of the DFS by considering the presence of the client-side caches, global cache and speculative processing. The main advantages of our algorithms are (i) Reduction in read access time (ii) Write operations do not perform caching and do not require the execution of cache invalidation or synchronization protocols. Our performance evaluation results indicate that the proposed algorithms perform better than the algorithm which does not use caching and speculative processing.

Peer-review under responsibility of scientific committee of 2nd International Symposium on Big Data and Cloud Computing (ISBCC'15)

Keywords: Distributed System; Speculation; Performance;

1. Introduction

A large number of web applications are being deployed in cloud computing systems which provide scalable infrastructure to store and process Big Data generated by various organizations. Distributed file system (DFS) is one of the important components of a cloud computing system used for storing and sharing Big Data among its authorized users. We can say that DFS is the main Big Data storage system used by many Big Data applications. More frequently the users of the web applications perform read operations on the DFS and less frequently write operations are carried out on the DFS. So, improving the performance of the read operations in the DFS is one of the important research issues in the Big Data domain.

Email addresses: ragu_savi@yahoo.com (Thirumalaisamy Ragunathan), sudheer.itdict@gmail.com (Sudheer Kumar Battula), rathnagurram@gmail.com (Rathnamma Gopisetty), burujula1971@gmail.com (B. RangaSwamy), geethanjali.sku@gmail.com (N. Geethanjali)

Peer-review under responsibility of scientific committee of 2nd International Symposium on Big Data and Cloud Computing (ISBCC'15)
doi:10.1016/j.procs.2015.04.050

Many pre-fetching and caching techniques are proposed in the literature for reducing the read access time in the DFS. Pre-fetching techniques are used for fetching the contents of a file or a block in advance into the cache based on the application access patterns. Pre-fetching can improve the performance of the read operations carried out on the DFS. This performance enhancement is dependent on the accuracy of the results of the analysis of log maintained in the DFS. A speculation-based approach is proposed in [4] for improving the performance of read operations in the DFS. Speculation-based approach is proposed in the literature for database systems to improve the throughput performance of transaction processing.

Disk devices are slow in comparison with main memory devices and hence reducing the disk I/O for improving the performance of the file system is one of the research issues in the literature. Caching and pre-fetching techniques reduce the disk I/O traffic by permitting the client programs to read the requested data from the cache maintained in the local system provided the data is available there. Most of the DFSs follow client-side caching techniques for improving the performance of the DFS. In the literature, client-side caching techniques like collaborative caching or cooperative caching are discussed to improve the performance of read operations in the DFS. The bottleneck with these approaches is that the application programs getting executed in the client systems have to wait for the server system to read the data from the disk devices attached to it. These client-side caching techniques proposed in the literature can improve the performance of the read operations the DFS. In this paper, we propose two algorithms based on speculative processing and client-side caching techniques.

Modern DFSs like Hadoop Distributed File System consists of one name node where the meta data is maintained and one or more data nodes. The data nodes are used for storing data and for executing user application programs. This DFS follows replication techniques for improving the performance and for implementing reliable file system features. Hadoop DFS maintains three copies of each file in the data nodes [7]. In this paper, we have proposed novel read algorithms for improving the performance of read operations in the DFS. The proposed algorithms perform speculative executions by reading the data from the local client cache and one from the global cache which is maintained separately and accessed by all the data nodes. Based on the time stamp verification process, one of the speculative executions may be considered. If the time stamps are not matching then the data will be read from one of the data nodes.

The main advantages of our algorithms are (i) Reduction in read access time (ii) Write operations do not perform caching and do not require the execution of cache invalidation or synchronization protocols. We have evaluated the performance through simulation and mathematical modeling and the results indicate that the proposed algorithms perform better than the algorithm which does not use caching and speculative processing.

This paper is organized as follows. In the next section, we describe the techniques discussed in the literature for improving the performance of the DFS. In section 3, we discuss our proposed algorithms in detail. In section 4, we have done the detailed performance evaluation of the algorithms. Section 5 concludes the paper.

2. Related Work

We discuss regarding the techniques discussed in the literature for improving the performance of the DFS. First, we discuss regarding client-side caching techniques which have been proposed in the literature for improving the performance of distributed file systems.

Cooperative caching technique is discussed in [1]. In this technique, the caches maintained in the client nodes will cooperate to satisfy the read access requests given by the application programs getting executed in one of the client nodes. A decentralized cooperative caching algorithm was proposed in [6] which can avoid the single-point of failure. Collective caching technique is discussed In [3]. This technique considers all the tasks that execute the same application program as a single client to the server. This allows the data available in the cache to be managed collectively within the client processes and makes every caching operation is known to all the tasks. The disadvantage of this approach is that the access pattern of individual task may be lost after the client file access requests are aggregated. So, this type of operation has a negative impact on the performance of the caching algorithm.

An aggressive, proactive mechanism is presented in [5]. In this technique, the application programs should provide hints which will be used for performing I/O in a parallel manner. In [2], locality-aware co-operative caching protocol is discussed. It is used to effectively predict cache utilization and the probability of data reuse.

A speculation-based method is discussed in [4]. In this technique, first, the data will be read from the local cache (if it is available there) and one speculative execution will be started. Then, the time stamp value is obtained from the server system and if the time stamp matches with the time stamp value of the local cached copy then the speculative execution can continue its work; otherwise that execution is stopped and the data will be read from tghe server system's disk. This type of processing can improve the performance by reducing the disk Input/Output operations.

The Hadoop DFS (HDFS) is developed by Hadoop which is a open-source project [7]. HDFS is designed to install on commodity hardware. HDFS is mainly used for applications which require data intensive operations. Replication is the technique used in HDFS for implementing fault tolerance feature. HDFS does not use caching and speculative processing.

3. Proposed Caching and Speculation-Based Algorithms

In this section we discuss regarding the proposed caching and speculation-based algorithms.

3.1. Speculative processing

In speculative processing, a task will be carried out before it is known whether that task is required or not. Then based on conditions the outputs of the completed task will be accepted. This type of processing will reduce the waiting time and can improve the performance. But, this type of processing requires more processing power from the computer system. Modern multi-core processors have abundant computing power and hence carrying out speculative processing is not a problem in these systems. Speculative processing is used in pipeline processors. Speculative executions are performed in the transaction processing systems for improving the throughput performance.

3.2. Proposed Algorithms

Assumptions:

We consider a DFS which consists of one name node and multiple data nodes and all these nodes are connected through a network. The name node stores the meta data and the data nodes store the data and execute user application programs. We also consider that data nodes maintain local caches in the main memory. Also, a global cache is maintained in a separate system. Caching is done only during reading and writing operation does not require caching activity. Note that, cache invalidation or synchronization protocol is not used in our proposed algorithms in order to avoid communication overhead. We consider that a client program is getting executed in the data nodes which will communicate with the name node and other data nodes. We also assume that entire file contents are cached in the local and global caches.

First, we discuss the algorithm which uses local and global caches for serving the read requests generated in the DFS.

Caching-based Algorithm:

/* A client application program AP executes in the data node DN1 has to read a File Fl1 */

```
Timestamp t1=getNameNodeTimeStamp(File Fl1)
Timestamp t2=getLocalCacheTimeStamp(File Fl1)
Timestamp t3=getGlobalCacheTimeStamp(File Fl1)
if t1 != null then
   if t2 != null AND t1 == t2 then
      Read Fl1 from local cache
```

else if *t3!=null AND t1 == t3* **then**
Read Fl1 from global cache
Copy Fl1 to local cache
else
Addresses of data nodes where Fl1 is available are obtained and Fl1 is read from the nearest data node (ND).
Copy Fl1 from ND to global cache and local cache
end if
else
File not found
end if

Next, we discuss the algorithm which uses local and global caches and speculative processing for serving the read requests generated in the DFS.

Speculative Processing and Caching-based Algorithm:
/* A client application program AP executes in the data node DN1 has to read a File Fl1 */
Timestamp t1=getNameNodeTimeStamp(File Fl1)
Timestamp t2=getLocalCacheTimeStamp(File Fl1)
Timestamp t3=getGlobalCacheTimeStamp(File Fl1)
Create a Speculative Excecution(SP1) and SP1 reads Fl1 (if available) from local cache
Create a Speculative Excecution(SP2) and SP2 reads Fl1 (if available) from global cache and copy Fl1 to local cache
if *t1!=null* **then**
if *t2!=null and t2 == t1* **then**
Terminate SP2 and Wait for SP1 to complete
else if *t3!=null and t3 == t1* **then**
Terminate SP1 and Wait for SP2 to complete
else
Addresses of data nodes where Fl1 is available are obtained and Fl1 is read from the nearest data node (ND1).
Copy Fl1 from ND1 to global cache and local cache
end if
else
File not found
end if

4. Performance Evaluation

We have considered that the block for the DFS as 4 KB and the average communication delay (ACD) required for transferring 4 KB of data from a remote data node to the local data node as 4 ms and for transferring time stamp and meta data information as 0.125 ms based on the recent analysis by considering the switched local area network. The average time required to access a data block (4 KB size) from the disk storage system is 12 milliseconds by considering the latest seagate disk storage devices. We have considered that the average time required to access 4 KB of data block from the main memory as 0.005 ms by considering latest DDR4 dynamic random access memory technologies and the time required to read 4 KB of data from the remote memory as 4.01 ms. Let us consider that local cache hit ratio as c and global cache hit ratio as gc.

Average read access time for a 4 KB data block of DFS without caching and speculative processing= reading 4 KB data block from a specific data node from the disk + reading 4 KB data block from the main memory of source data node + transferring the 4 KB data block to the client data node + copying the data in the client node's main memory.

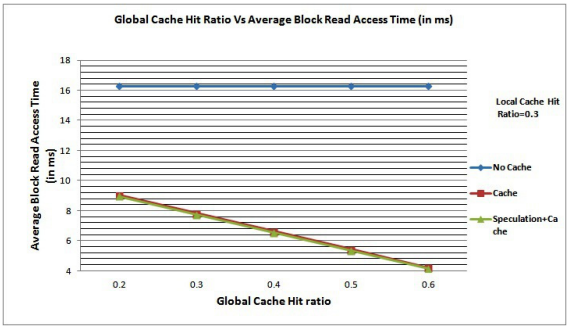

Fig. 1: Global Cache Hit Ratio Vs Average Block Read Access Time(Local Cache Hit Ratio is 0.3)

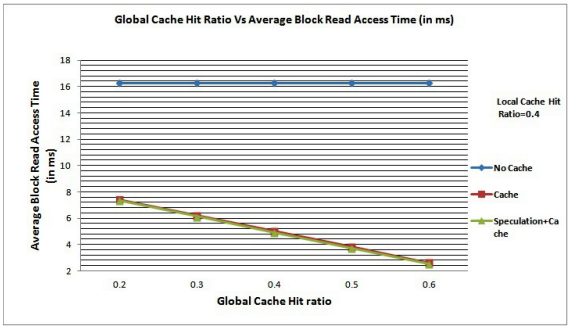

Fig. 2: Global Cache Hit Ratio Vs Average Block Read Access Time(Local Cache Hit Ratio is 0.4)

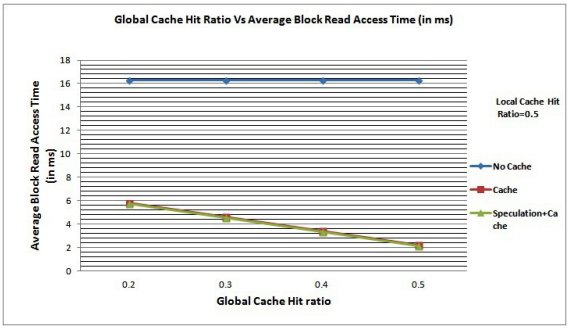

Fig. 3: Global Cache Hit Ratio Vs Average Block Read Access Time(Local Cache Hit Ratio is 0.4

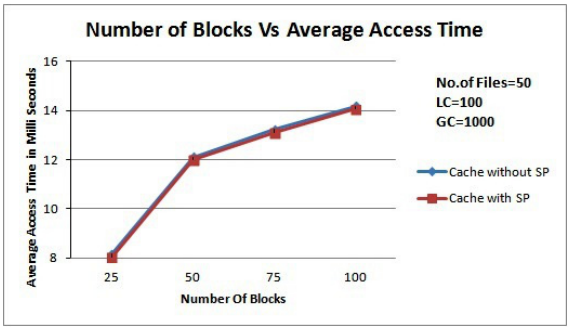

Fig. 4: Number of Blocks Read Vs Average Block Read Access Time)

Average read access time for a 4 KB data block of DFS by considering local and global caching= *c* * (time required to access the local memory (local cache) + time required to access name node to collect time stamp) + *gc* * (time required to access the local memory (local cache) + time required to access name node to collect time stamp + time required to transfer the time stamp value form the node where global cache is present + time required to transfer the data from remote data node's memory (global cache) to local data node's memory) + (1-*c*-*gc*) * (time required to access the local memory (local cache) + time required to access name node to collect time stamp + time required to transfer the time stamp value form the node where global cache is present + reading 4 KB data block from a specific data node from the disk + reading 4 KB data block from the main memory of source data node + transferring the 4 KB data block to the client data node + copying the data in the client node's main memory)

Average read access time for a 4 KB data block of DFS by considering local and global caching and speculative processing= *c* * (time required to access name node to collect time stamp) + *gc* * (time required to access name node to collect time stamp + (time required to transfer the data from remote data node's memory (global cache) to local data node's memory - time required to access name node to collect time stamp) + (1-*c*-*gc*) * (time required to access name node to collect time stamp + reading 4 KB data block from a specific data node from the disk + reading 4 KB data block from the main memory of source data node + transferring the 4 KB data block to the client data node + copying the data in the client node's main memory)

The results of mathematical analysis are shown in figures 1, 2 and 3. In Fig. 1, we have fixed the *c* value as 0.3 and varied *gc* values from 0.2 to 0.6 and observed the average block read access time performance of the proposed algorithms. We can observe that for all *c* and *gc* values, the proposed speculation and caching-based algorithms perform better than the algorithm which does not use caching and speculation. In Fig 2. we have analysed the performance difference between speculation and caching-based algorithm and pure caching-based algorithm. Approximately 100 micro seconds difference is there between speculation and caching-based algorithm and pure caching-based algorithm.

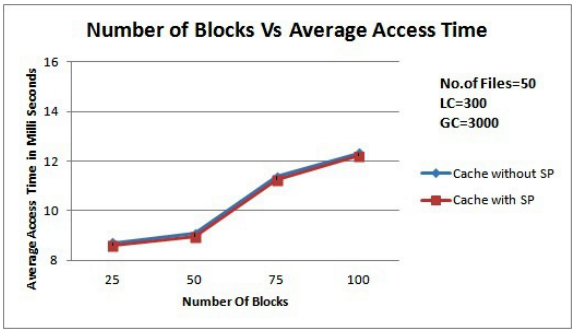

Fig. 5: Number of Blocks Read Vs Average Block Read Access Time)

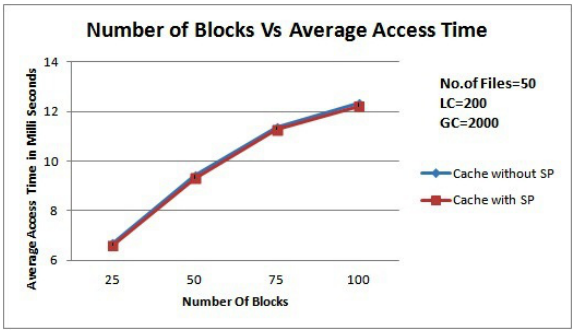

Fig. 6: Number of Blocks Read Vs Average Block Read Access Time)

The results of simulation experiments are shown in the figures 4, 5 and 6. We observe that the caching and speculation-based caching algorithms are performing better than the one which does not support caching and speculative processing. Overall, the results of both mathematical analysis and simulation experiments indicate that the proposed speculative processing and caching-based algorithms perform better than the algorithm which does not use caching and speculative processing.

5. Conclusion and Future Work

Distributed file system is used as the main component in Big Data storage systems. In this paper, we have proposed two novel read algorithms for improving the performance of the read operations in the distributed file system based on caching and speculative processing. The main advantages of our algorithms are (i) Reduction in read access time (ii) Write operations do not perform caching and do not require the execution of cache invalidation or synchronization protocols. The results of our performance evaluation indicates that the proposed algorithms require less read access time than the algorithm which does not use caching and speculative processing. In future, we plan to implement the algorithms in the Hadoop Distributed File System to prove their efficiency.

References

[1] M. D. Dahlin, R. Y. Wang, T. E. Anderson, and D. A. Patterson. Cooperative caching: Using remote client memory to improve file system performance. In *Proceedings of the 1st USENIX Conference on Operating Systems Design and Implementation*, OSDI '94, Berkeley, CA, USA, 1994. USENIX Association.

[2] S. Jiang, F. Petrini, X. Ding, and X. Zhang. A locality-aware cooperative cache management protocol to improve network file system performance. In *Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on*, pages 42–42, 2006.

[3] W.-k. Liao, K. Coloma, A. Choudhary, L. Ward, E. Russell, and S. Tideman. Collective caching: application-aware client-side file caching. In *High Performance Distributed Computing, 2005. HPDC-14. Proceedings. 14th IEEE International Symposium on*, pages 81–90. IEEE, 2005.

[4] E. B. Nightingale, P. M. Chen, and J. Flinn. Speculative execution in a distributed file system. In *Proceedings of the Twentieth ACM Symposium on Operating Systems Principles*, SOSP '05, pages 191–205, New York, NY, USA, 2005. ACM.

[5] R. H. Patterson, G. A. Gibson, E. Ginting, D. Stodolsky, and J. Zelenka. Informed prefetching and caching. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, SOSP '95, pages 79–95, New York, NY, USA, 1995. ACM.

[6] P. Sarkar and J. Hartman. Efficient cooperative caching using hints. In *Proceedings of the Second USENIX Symposium on Operating Systems Design and Implementation*, OSDI '96, pages 35–46, New York, NY, USA, 1996. ACM.

[7] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1–10. IEEE, 2010.